HANDLING COLLISIONS DURING SYNCHRONIZATION OF DATA BETWEEN CLIENT AND SERVER COMPUTERS

Field of the Invention

This invention generally relates to synchronization of data maintained in a central storage with changes made in downloaded copies of the data, and more specifically, to synchronization of the data on a server in regard to changes made in the data on client computers that are periodically connected to the server, to resolve conflicts in the data occurring as a result of the changes.

Background of the Invention

There are numerous situations in which it is important to synchronize data that can be downloaded from a server and modified by more than one person or on more than one computer. Each person having access to the data can make changes that may be inconsistent with each other, causing errors in the data when the versions of the data that include these inconsistent changes are uploaded to the server. For example, a person who has downloaded data may add elements to the data and upload the modified data to the server. However, those modifications may be overwritten when another person uploads a different version of the data to the server. Relatively complicated software products have been developed to prevent such scenarios from occurring. Such software products are often used by groups of programmers who are cooperatively developing software to avoid conflicts as different portions of complex software code are written and then modified by different program developers. For example, Microsoft Corporation includes a Visual Sourcesafe™ program with its software development suite, Visual Studio™ that

enables developers to "check out" source code files, and when the files are "checked in," merges changes made by all of the users who have checked out the files, by comparing all versions of each file. However, there are many instances when a simpler approach to preventing conflicts and synchronizing the data modified by multiple users that does not merge changes by different users would be beneficial.

While many other types of data might benefit from a simple program to ensure synchronization of the data on the server and the data downloaded and modified by a plurality of users, an excellent example of such an application arises in connection with data stored on Microsoft Corporation's Encarta Class Server (ECS), which starting with version 2.0, will be sold under the trademark Class Server™. The ECS is a client/server product that was developed for use in kindergarten through twelfth grade schools (K-12) to manage distribution, collection, and grading of electronic school work, or "assignments." Students access the ECS using a Web interface. Teachers access ECS using the ECS teacher client (TC), which is also known as Class Server-Teacher. Using TC, a teacher can assign an assignment to students; once the assignment has been completed by students, the teacher can grade each student's electronic work product (called a "paper") for that assignment. Finally, students can view their graded papers. More information about the ECS is available on the Internet at the Web site http://ECS.msn.com/.

Teachers can use TC to create assignments and grade papers remotely over the Internet (e.g., while the teacher is working at home), or can work offline, i.e., while not connected to an intranet or the Internet. When the teacher reconnects to the network, it is important that the TC synchronize the upload to the ECS of any changes (to assignments and papers) that the teacher made, and download any new or modified assignments as well as papers completed by students. The ECS must allow teachers to collaborate. For example, while working offline at home, a teacher should be able to create an assignment, while a teaching assistant grades papers at school. When either the teacher or teaching assistant clicks a synchronize button in the TC, that person should receive the changes made by the other during the synchronization, and changes made by the teacher should then be uploaded to the ECS.

Accordingly, it is important that assignments and papers be synchronized between client and server, and that collisions between modified data be detected and resolved. Collisions can occur if two teachers change the same assignment at the same time. The synchronization process should be incremental, requiring only data transfer of information that has changed since the last synchronization occurred. A program that can provide such functionality will thus be broadly applicable to other types of data and other applications in which data synchronization is important

Summary of the Invention

The present invention is directed to a method for maintaining synchronization of data stored on a server where the data are accessible by a plurality of clients. The clients can be coupled in communication with the server or can operate offline. When the clients are coupled to the server, they can download data from the server so that a user can modify the data on the client. Changes to the data made on the client can also be uploaded to the server. The data include a plurality of nodes; a node is at an atomic level, i.e., a node is the smallest data element that can be synchronized as a unit in accord with the present invention. In the example that is discussed below, nodes correspond to class assignments that are created and can be modified by teachers using client computers, either at home or at school. However, the present invention is not limited to the specific exemplary application discussed below, and it should be understood that nodes can be other kinds of data. Also, in the exemplary application discussed below, nodes are grouped together for each class taught by a teacher. However, in a more general sense, the term "class" as used herein has a much broader meaning and is intended to refer to an atomic level within which nodes are grouped for purposes of synchronization, i.e., a class is a set of nodes to which synchronization between a server and client computers is applied, in accord with the present invention.

A version identifier (positive integer) is assigned to a class comprising the nodes on a server. More importantly, the data comprising the nodes for a class stored on the server has a version identifier that is incremented each time any node of the data for that class is modified on the server (preferably, before data on the server are actually changed

to include the modified node). The new version identifier class (that was just incremented) is then associated with the modified node in the data for the class on the server. In addition, a "node deleted" version identifier is associated with the data in the class, and has the value of the version identifier that was assigned to the data when a node was last deleted on the server.

When synchronizing data on the client and data on the server for a class, nodes that have been modified on the server since the nodes were previously downloaded by a client will be downloaded to the client to update the client's cache of the data. Also, nodes that were previously downloaded from the server by a client and subsequently modified on the client will be uploaded to the server, so that the changes in the modified node are then available to the next client synchronizing with the server. Similarly, nodes that were newly created by a client are uploaded to the server.

During the synchronization process, the present invention provides for detecting any "collision" occurring between a node that has just been downloaded from the server to a client and a corresponding node that was previously downloaded and modified on the client. A collision occurs when a node was modified by two clients, both starting from the same version of the node downloaded from the server, and an attempt is made to synchronize with server by uploading the nodes that were modified. The collision occurs during the second client's attempt to synchronize. A "proactive collision" is detected during the download from the server of a node that was modified previously by another party, where the downloaded node corresponds to a node that was modified on the client since a previous synchronization by the client occurred. The differences in the version identifiers of the two corresponding nodes enables the client to detect a proactive collision. If a proactive collision is detected, the user of the client is alerted. A reactive collision is detected during an upload to the server of a node that was modified by a first client and is typically caused by a second client synchronizing at about the same time as the synchronization process of the first client is occurring.

The method thus also provides for detecting any reactive collision between corresponding modified nodes that were separately modified on two clients and which

are being uploaded at about the same time. Again, the reactive collision is detected based upon the version identifiers associated with the nodes that are being uploaded. If a reactive collision is detected on the server, then the method causes the synchronization process to be repeated by one of the clients. During the repeated synchronization, a proactive collision is detected, so that the user of the one client is alerted to the problem.

More specifically, a reactive collision occurs during upload of a node by a second client after a first client has completed uploading of a corresponding modified node at about the same time as the second client. During the second client's upload, the server notices that the original version identifier of the node being uploaded by the second client is different than the server's current version identifier, which indicates to the server that a modified version of the node has been uploaded to the server since the time that the second client downloaded the node. The server then aborts the second client's upload process, and the second client is caused to restart the synchronization process so that the collision can be detected as a proactive collision and handled appropriately by the user of the second client.

Before each download of nodes from the server to the clients occurs, the version identifier for the class on the client is transferred to the server, since this version identifier indicates when the previous synchronization of the client with the server occurred. Any nodes for which the version identifier associated therewith indicates the node on the server is a later version than the corresponding node on the client are downloaded to the client. During the download, the method automatically overwrites each node that was not yet modified on the client, with a corresponding node downloaded from the server, and deletes each node on the client that is no longer on the server.

Each client preferably maintains a cache in which are stored the latest version of nodes most recently downloaded from the server. Each client also maintains a storage containing all nodes that have modified on the client, but not yet uploaded to the server.

If a proactive collision is detected on a client, a user can choose either to overwrite the modified node on the client with the corresponding node just downloaded from the server, or to upload any modified node from the client to the server, overwriting

any corresponding node on the server. In a current preferred embodiment, the decision is made in regard to all proactive collisions identified during the current synchronization.

Another aspect of the present invention is directed to a memory medium having machine instructions for performing the steps generally as described above in regard to the method. Yet another aspect of the present invention is directed to a system for maintaining synchronization of data. The system includes a server computing device and client computing devices. The server computing device and each client computing device includes a memory in which machine instructions are stored and a processor that executes the machine instructions, carrying out functions that are generally consistent with the steps of the method described above.

Brief Description of the Drawing Figures

The foregoing aspects and many of the attendant advantages of this invention will become more readily appreciated as the same becomes better understood by reference to the following detailed description, when taken in conjunction with the accompanying drawings, wherein:

FIGURE 1 is a functional block diagram of a generally conventional personal computer that is suitable for use as either the client or the server computing device for implementing the present invention;

FIGURE 2 is a functional block diagram of a first exemplary system on which the present invention is implemented;

FIGURE 3 is a functional block diagram of a second exemplary system on which the present invention is implemented;

FIGURE 4 is a functional block diagram of a third exemplary system on which the present invention is implemented;

FIGURE 5 is a functional block diagram illustrating the logical steps carried out when synchronizing the data on a client with the data stored on the server in accord with the present invention; and

FIGURE 6 is block diagram illustrating details of the functional software components employed on each client and the server of an exemplary system on which the present invention is implemented.

Description of the Preferred Embodiment

Computer for Implementing the Present Invention

FIGURE 1 and the following discussion related thereto are intended to provide a brief, general description of a suitable computing environment in which the present invention may be implemented. This invention is preferably practiced using one or more computing devices functioning as a server, which are coupled to each client computing device or other remote computing device by a communications network. Both the server and the client computing devices will typically include the functional components shown in FIGURE 1. Although not required, the present invention is described as employing computer executable instructions, such as program modules that are executed by the server and by the client computers to enable synchronization of data exchanged. Generally, program modules include application programs, routines, objects, components, functions, data structures, etc. that perform particular tasks or implement particular abstract data types. Also, those skilled in the art will appreciate that this invention may be practiced with other computer system configurations, particularly in regard to the client, including handheld devices, pocket personal computing devices, digital cell phones adapted to execute application programs and to wirelessly connect to a network, other microprocessor-based or programmable consumer electronic devices, multiprocessor systems, network personal computers, minicomputers, mainframe computers, and the like. As indicated, the present invention may also be practiced in distributed computing environments, where tasks are performed by one or more servers in communication with remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to FIGURE 1, an exemplary system for implementing the present invention includes a general purpose computing device in the form of a personal computer 20 that is provided with a processing unit 21, a system memory 22, and a system

bus 23. The system bus couples various system components, including the system memory, to processing unit 21 and may be any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system (BIOS) 26 containing the basic routines that are employed to transfer information between elements within computer 20, such as during start up, is stored in ROM 24. Personal computer 20 further includes a hard disk drive 27 for reading from and writing to a hard disk (not shown), a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31, such as a CD-ROM or other optical media. Hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical disk drive interface 34, respectively. The drives and their associated computer readable media provide nonvolatile storage of computer readable machine instructions, data structures, program modules, and other data for personal computer 20. Although the exemplary environment described herein employs a hard disk, removable magnetic disk 29, and removable optical disk 31, it will be appreciated by those skilled in the art that other types of computer readable media, which can store data that are accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks (DVDs), Bernoulli cartridges, RAMs, ROMs, and the like, may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, or in ROM 24 or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37, and program data 38. A user may enter commands and information into personal computer 20 through input devices such as a keyboard 40 and a pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input/output (I/O) devices are often connected to processing unit 21 through an I/O interface 46 that is coupled to system bus 23. The term I/O interface is intended to

encompass interfaces specifically used for a serial port, a parallel port, a game port, a keyboard port, and/or a universal serial bus (USB), and other types of data ports. A monitor 47, or other type of display device, is also connected to system bus 23 via an appropriate interface, such as a video adapter 48, and is usable to display application programs, Web pages, and/or other information. In addition to the monitor, the server may be coupled to other peripheral output devices (not shown), such as speakers (through a sound card or other audio interface, not separately shown), and printers.

As indicated above, many aspects of the present invention are preferably practiced on a single machine functioning as a server; however, personal computer 20 will normally operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 49, which may be one of the client computers exchanging data over the network. Remote computer 49 may alternatively be a server, a router, a network PC, a peer device, or a satellite or other common network node, and typically includes many or all of the elements described above in connection with personal computer 20, although only an external memory storage device 50 has been illustrated in FIGURE 1. The logical connections depicted in FIGURE 1 include a local area network (LAN) 51 and a wide area network (WAN) 52. Such networking environments are common in offices, enterprise wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, personal computer 20 is connected to LAN 51 through a network interface or adapter 53. When used in a WAN networking environment, personal computer 20 typically includes a modem 54, or other means such as a cable modem, Digital Subscriber Line (DSL) interface, or an Integrated Service Digital Network (ISDN) interface, for establishing communications over WAN 52, which may be a private network or the Internet. Modem 54, which may be internal or external, is connected to the system bus 23 or coupled to the bus via I/O device interface 46; i.e., through a serial port. In a networked environment, program modules depicted relative to personal computer 20, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections

shown are exemplary and other means of establishing a communications link between the computers may be used, such as wireless communication and wideband network links.

Exemplary Application of the Present Invention

Although it should clearly be understood that the present invention is not in any way limited by the following example, an initial application, which clearly illustrates how it is applied, is described below. This example is particularly pertinent because the present invention solves a troublesome problem that would otherwise be experienced by users of client computers in a client-server network over which data are accessed. Specifically, the present invention was developed to address a problem regarding synchronization of data for electronic schoolwork or assignments. Microsoft Corporation's ECS is a client-server product for use in kindergarten through twelfth grade (K-12) schools for managing, distributing, collecting, and grading of assignments. Each assignment, which is a node of the data maintained by the ECS, may have an attached unit of educational content called a "learning resource" (LR). Teachers and students typically access the ECS server over a LAN or over the Internet using a conventional Web interface, such as Microsoft Corporation's Internet Explorer™ browser program. In addition, teachers access the ECS using the TC, as noted above. When connected to the network using the TC, a teacher can assign an assignment to students in the class directed by the teacher, and once the assignment is completed by the students, the teacher can grade each student's electronic work product online (while connected to the ECS server over a LAN or Internet connection), or while working offline (while disconnected from the LAN or Internet). In addition, the students can view their graded work over the network using their browser program.

To provide flexibility for this electronic class work paradigm, teachers periodically synchronize their TECS with the server and then may choose to work away from school. For example, while at school, the teacher can use a TC at the school to carry out modifications, or create assignments, or grade papers, or alternatively do the same work while connected to the server over the Internet, while at home. Furthermore, having downloaded data for existing assignments or other types of nodes related to a

class, a teacher can also work offline, while not connected either to the intranet or to the Internet. When the teacher next connects the TC to the ECS over a network, the TC will synchronize by downloading modified nodes and any new nodes from the ECS and uploading to the ECS server any changes that might have been made to assignments or papers by the teacher.

The ECS system enables teachers to collaborate on the work for a class, which is important since two or more teachers may work at creating assignments for a specific class. Thus, for example, while working at home, a teacher may create an assignment while concurrently, a teaching assistant may be grading papers for another assignment at school. When either the teacher or the teaching assistant clicks a synchronize button in their respective TC, they will receive a download of any changes previously uploaded by the other in the data related to the class.

In order for the system described above to work properly, it is necessary to maintain synchronization between data modified on the ECS and changes made to data that have been downloaded or created on a plurality of TECS. The present invention ensures that data nodes, such as assignments and papers, are synchronized between the clients and the server and detects any collisions of the nodes that have been modified, for example, in the event that two individuals changed corresponding assignments or other kinds of nodes. To simplify and improve the performance of this synchronization process, the present invention is incremental, because it attempts to only require the transfer of data for information that is changed since a previous synchronization between the ECS and a TC last occurred.

For each school, the ECS maintains a collection of zero or more classes for which data related to assignments and papers are included. Each class (Cls) includes a collection of zero or more assignments, and information comprising an assignment (As) is called an "As node." In addition, each class typically has one or more teachers (it is contemplated that the present invention might also be applied to an electronic teaching paradigm that does not require any teacher be assigned to a class). Each teacher runs the TC software on a local copy or cache of the data for one or more classes that has been

downloaded to the TC. This copy preferably includes the As nodes for each class that the teacher teaches. Periodically, a teacher uses the synchronization capability of the TC to first download from the ECS any modifications made by other teachers of the same classes, or made by the same teacher on another computer running the TC, and to upload modifications to the data made by the teacher on a TC to the ECS.

FIGURES 2, 3, and 4 illustrate several different embodiments of network configurations in which the present invention is employed to maintain synchronization of data that is accessible at different locations by a plurality of different client computers. FIGURE 2 illustrates a configuration 100 in which a school 102 includes a server computer 118 that runs both ECS and Microsoft Corporation's SQL Server 2000™ database program 120. A plurality of teacher personal computers 104 and 106 (only two shown) are connected to the server computer running ECS, over an intranet 112 through a proxy server 114. Proxy server 114 is coupled to the server computer running the ECS through an extranet 116. Also connected to the ECS via the intranet are a plurality of student personal computers 108 and 110. The student personal computers have the right to access but not change assignments and enable students to prepare electronic papers for submission to the teachers in response to the assignments provided by the teachers. A router 122 connected through extranet 116 to the server computer running the ECS is also coupled to Internet 124 so that data can be downloaded or uploaded from server computer 118 over Internet 124. Thus, at locations outside school 102, such as at their respective homes, teachers can use teacher personal computers 126 and 128 and students can use student personal computers 130 and 132 to access data stored on server computer 118 over Internet 124 and to upload data to the ECS on the server computer. An advantage of embodiment 100 is that teachers and students at school 102 can readily access and use the ECS without being connected to Internet 124. Although not shown in any of the embodiments of FIGURES 2-4, an administration personal computer, which is running the ECS Web based administrator user interface, enables information about teachers, students, and classes to be added, removed, or updated in the data stored on server computer running the ECS.

Referring to FIGURE 3, an embodiment 100′ illustrates a different configuration in which each school has a specific one of server computers 136-140 for running the ECS, but each of the server computers are maintained at an Application Server Provider (ASP) data center 134 separate from the school, so that the school doesn't have to maintain an ECS server. In this embodiment, teacher and students must always use a personal computer to connect to the ECS over Internet 124 (or some other external network) to access the data maintained by the ECS in the SQL 2000 database 138 of the school, regardless of whether the teachers and students are at home or at school 102. Thus, server computer 136 includes an SQL 2000 142 database to maintain the data for school 102, while server computer 140 includes another SQL 2000 database to maintain the data for a different school *N*. One advantage of this embodiment is that it is relatively easy for the ASP data center to set up the ECS for a specific school, but the embodiment does not scale up as additional schools are added, as well as embodiment 100.

In FIGURE 4, an embodiment 100″ is illustrated in which an ASP data center 134′ divides the collection of schools that are served into clusters, with *N* schools per cluster. As shown in this example, a cluster 146 includes a server 152, a server 154, and a total of up to *M* servers, up through server 156 that execute the ECS. These servers are in turn connected to access *N* databases 162 and 164 running under SQL 2000 160. A Windows Load Balancing Server (WLBS) 158 is used to distribute requests for data directed to the ECS to selected ones of servers 152, 154, and 156, so that the data processing load is distributed among the collection of servers in cluster 146. Similarly, additional clusters 148 and 150 are also maintained at ASP data center 134′, each cluster servicing a different one of up to *N* schools. It should be noted that each school's data are contained in a single one of databases 162-164, so that the data for each of the *N* schools are kept separate and not co-mingled. Also, the *M* servers running the ECS can be increased or decreased in value as needed, on the fly, based on demand. Any of these servers can thus service a request from any of the *N* schools, because the servers are totally stateless.

FIGURE 5 illustrates the software modules that are employed on an administrative client 300, a TC 340, and a student client 350, all of which are coupled in

communication with a server computer 320 on which the ECS is executed. Administrative client 300 includes an ECS administrative interface 302 downloaded as Dynamic Hypertext Markup Language (DHTML) over Hypertext Transport Protocol (HTTP) from the ECS. Also installed on the administrative client is a browser program 304, which typically operates in a graphic user interface operating system 306, such as Microsoft Corporation's Windows 95™ (or later) operating system.

TC 340 executes the ECS teacher user interface that is installed as a DHTML application 342 and which includes a clienthelp10.dll 344 (i.e., a dynamic link library) to provide help assistance to the TC application. The ECS teacher user interface communicates using DHTML over HTTP and the clienthelp10.dll module communicates with an Internet server application programming interface (ISAPI) filter 322 on the ECS, using XML and Multipurpose Internet Mail Extensions (MIME) over HTTP. Also coupled to this filter using HTML over HTTP is an ECS student user interface 352 that is on student client 350. The ECS student user interface is provided using downloaded HTML over HTTP. Teacher client 340 also includes a browser program 346 and preferably, has a graphic user interface operating system 348 like that on the administrative client.

Student client 350 also includes a browser program 354 and a graphic user interface operating system 356, which can be either a PC-compatible system or Apple Computer Corporation's Macintosh™ operating system, version 8.5 or later. The ECS on the server computer also includes an ISAPI module 324, a schdta10.dll 326, and a general server module 328. Schdta.dll module 326 is coupled to the SQL Server 2000 or a corresponding Microsoft Data Engine (MSDE) 330, which is under the control of server 332. Data are thus exchanged between the TECS (only one shown) and the student clients (only one shown) over the HTTP connection with the ECS using the modules included in server computer 320, as shown in FIGURE 5.

Data Synchronization

There are two types of collision detection that must be performed to enable data on the ECS to be synchronized with data on the TECS. A data collision, which is

sometimes called a "conflict" in other applications, occurs when an assignment (or other type of data node) is modified by two users, or by the same user on two different computers, and an attempt is then made to synchronize with the server running the ECS. When corresponding nodes of the data are thus changed by two or more clients and a synchronization causes a changed node to be uploaded to the server running the ECS, the resulting collision that occurs when another client attempts to synchronize must ultimately be resolved by a user. The two types of collisions, which are clearly explained below, are "a proactive collision" and "a reactive collision."

FIGURE 6 illustrates the logical steps implemented in accordance with the present invention for maintaining synchronization of data for a single class. In the event that data for a plurality of classes must be synchronized, the steps illustrated in FIGURE 6 will simply be repeated for each class, as necessary. FIGURE 6 is divided by a vertical dash line that separates the steps that are carried out by the TC on the client computer, from those that are carried out on the server by the ECS. Once the synchronization process starts (e.g., when the teacher activates the synchronize button on the TC), at a step 200, the client sends a variable "TCls.SN" to the server to indicate the version identifier (or "sequence number") of the data (for that specific class) that the client currently has stored in its cache. At the time the system is initially set up, it is assumed that the client downloaded at least a first version of the data from the server and may have downloaded a number of updates, so that some version of the data on the server is already stored by TC in the cache on the client. In response to receiving this variable, in a step 202, the server sends the client assignment updates, file data, and a list of students currently in the class to which the data relates. (In a current embodiment, the TC synchronizes assignment node data first – in one iteration of the steps shown in FIGURE 6, and then synchronizes file data – in a second iteration of FIGURE 6.) More specifically, in this specific example, the server sends back to the client all DB.As nodes (assignments maintained in the database on the server) for which DB.As.SN (i.e., the version identifier for the assignment) is greater than the value received from the client for the variable T.Cls.SN. Accordingly, it should be apparent that only the assignment data

that have been updated since the data were last downloaded from the server by the client need to be provided to the client to update its data to correspond to the data stored on the server. Also included in the data provided to the client by the server at this step is a list As IDs (i.e., unique assignment identifiers) of the DB.As nodes that currently exist on the server; but, this list is only provided if assignments have been deleted on the server since the class version number was T.Cls.SN. This list enables the client to determine which DB.As nodes have been deleted.

Upon receiving the data from the server, in a step 204, the client merges the server-provided data into cache T, which is a mirror of information maintained on the server, and also into client store MT, which contains all information that has been modified on the client. Both T and MT are stored on the client computer's hard disk, and are also maintained in memory while TC is running. During this step, the data for each DB.As node that is downloaded is copied into T, overwriting the T.As node if it already exists there. In addition, if an MT.As node has not been modified on the client since the last synchronization occurred (i.e., if it is not flagged as being "dirty"), then each such DB.As node is copied into MT, thereby overwriting a corresponding MT.As, if that assignment node already exists. Otherwise, only the "server attributes" (specific portions of assignment data that are never modified on the client, except during initial assignment creation) are copied to MT.As. Also, if the server provided a list of AsIDs for the assignment nodes that are currently on the server since the previous synchronization by this client, the ID list is compared to the current list of assignments on the client, and those client assignments that are not on the list provided by the server are deleted from the assignments maintained in the client's cache T. At this point, the client's data are synchronized to reflect all changes of the data that have occurred on the server since the last synchronization by this client with the server.

Next, the client determines if a proactive collision is detected in a decision step 206. A brief example should help in understanding how a proactive collision occurs. In this example, it will be assumed that both a first teacher and a second teacher have synchronized their TECS to the ECS by carrying out steps 200-228. During the

synchronization, an assignment X is downloaded to the client computers of both the first teacher and the second teacher. It should be noted that the first and second teacher could actually be the same teacher at two different points in time, perhaps working at school in a first instance, and at home in a second instance. The first teacher edits the version of assignment X on the client computer, and the second teacher likewise edits assignment X on a different computer. Next, the first teacher carries out steps 200-228 to synchronize the TC on the client computer with the ECS, and in addition, uploads the modified version of assignment X to the server. Similarly, the second teacher subsequently executes steps 200, 202, and 204 to begin the process of synchronizing the TC with the ECS. However, upon receiving the data from the server in step 204, the TC on the second teacher's client computer recognizes that the assignment was modified on the server as a result of the first teacher's earlier uploading of the modified version of assignment X to the server since the time that the second teacher downloaded assignment X. Accordingly, if the second teacher uploads the modified version of assignment X that the second teacher created, it will collide with the modified version of assignment X that was created by the first teacher and which has already been uploaded to the server. As a result, in decision step 206, the TC for the second teacher determines that a proactive collision has been detected. This detection is carried out by having the TC compare the serial number or version identifier of assignment X that was modified by the first teacher, uploaded to the server. When the modified version of assignment X was uploaded to the server by the first teacher, the ECS on the server increments the serial number or version of the data on the server. Assignment X as modified by the first teacher will be downloaded to the second teacher, because it has a later version identifier than that of assignment X that was previously downloaded by the second teacher and modified by the second teacher. Specifically, since the DB.As.SN of assignment X as modified by the first teacher that was just downloaded is not equal to MT.As.SN for assignment X as modified by the second teacher, TC notifies that a proactive collision has occurred.

If a proactive collision is detected, a decision step 208 determines how the user has chosen to resolve the collision. (Currently, this decision is not enabled on a node-by-node basis, but instead, is applied to all proactive collisions that were determined in the current synchronization. However, the decision can be implemented on a node-by-node basis, for each proactive collision that is determined.) It should be noted that proactive collisions can be resolved in either of two ways. As indicated by a Path A leading to a step 210, the user may decide to keep the modified version in MT for an assignment identified as being involved in a proactive collision. Or alternatively, if following a Path B to a step 214, the user may decide to keep the version that was modified on the server and just downloaded, thereby overwriting the version that the user had modified in MT. In this case, as indicated in a step 216, the client versions of any node involved in a proactive collision that are in the MT on the client computer are deleted by overwriting with the corresponding versions that were modified on the server and were just downloaded. Following either step 210 or step 216 or as a result of a determination that a proactive collision was not detected, a step 212 calls for uploading the client assignment data to the server. This step corresponds to uploading any As node in MT that has been modified since the node was last updated (i.e., is flagged as being "dirty"), to the server. In addition, the client also sends the MT.As.SN, which identifies the serial number or version of the assignment that was in the client's MT. In an optional step 220, other client data can also be sent to the server including new-created assignments that were not previously downloaded from the server but have been produced by the teacher or other user of the client computer since the last upload of data.

A determination is made in a decision step 218 of whether a reactive collision has occurred. A reactive collision is detected if the value of MT.As.SN that was uploaded from the client in step 212 is not equal to the value of DB.As.SN for the data t on the server. In other words, decision step 218 determines if the modified data has a version identifier or serial number that is different than that of a corresponding assignment now stored on the server. The circumstances that lead to a reactive collision are somewhat

rare as will be evident from the following example, which explains how a reactive collision between corresponding nodes of data can occur.

In this example of a reactive collision, both a first teacher and a second teacher synchronize their respective TECS at about the same time, causing an assignment Y to be downloaded to their respective computers. Thereafter, both the first teacher and the second teacher separately modify the version of Y that was just downloaded to their respective client computers. Both teachers then initiate synchronization with the server, to upload their respective changes to Y. In this case, both teacher's TCs complete steps 200-206 at about the same time, and no proactive collision is detected by either TC because Y has not been uploaded yet. Then, the first teacher's TC completes step 212, and the server successfully completes steps 218-224. Finally, the second teacher's TC attempts step 212, but in step 218, the server detects a collision, because the version identifier of the second teacher's version of Y is not the same as the version identifier of the version of Y currently stored on the server (which, of course, is the version of Y that was just uploaded by the first teacher's TC). Thus, the server notifies the second teacher's TC of a reactive collision, and in response, the second teacher's TC restarts the synchronization process at step 200 — which this time results in a proactive collision being detected by the second teacher's TC, giving the second teacher the opportunity to resolve the collision, as discussed above.

It should be apparent that the decision about how to resolve a collision depends upon the relative authority of the person making the decision in regard to any newly downloaded node that is different than an earlier downloaded corresponding node that has been modified by the person. For example, a teaching assistant might be instructed never to override any data stored on the server with modifications made by the teaching assistant, since those modifications may have been made my a teacher who has greater authority to make changes in the data.

In the event that a reactive collision is not detected, which will most often be the case, in a step 222, the server updates or increments the value of DB.Cls.SN to indicate that a modification of the data stored on the server has occurred. Next, in a step 224, the

server merges the updated assignments just received from the client into the database. A step 226 provides for notifying the user of the client computer that a successful synchronization has just been completed. Next, in a step 228, a flag is set in the client's MT to indicate that a synchronization successfully transferred any modified assignments or other types of node to the server, and the logic then terminates.

Several details further clarify how the present invention is preferably implemented. For example, the version identifier can start at any value, but preferably starts with the value of zero for data initially stored on the server that have not yet been modified on the server. As noted above, anytime that data are transferred from a client to the server, causing the data on the server to be modified, the version identifier (serial number) associated with the data on the server is then incremented, providing an indication that the version of the data on the server has changed. The newly uploaded data are then associated with the new version identifier. It should also be understood that each client computer maintains two separate memory storages, including the client cache storage T that includes data mirroring the data on the server at the last synchronization, and the client store MT that includes assignments or other types of data nodes that have been modified on the client but have not yet been uploaded to the server. The flag that was set in step 226 of FIGURE 6 indicates to client store MT whether the modified nodes of the data have been transferred successfully to the server. The version number indicated by the serial number associated with the nodes of the data is important in the present invention, because it indicates a relative point on a timeline corresponding to when the nodes were updated on the server and the version of the nodes that were last downloaded from the server to the client. To avoid overflow, serial numbers are preferably represented as 64-bit integers.

Each object that is maintained on the server by the ECS, including classes, assignments, papers, students, etc., has a numerical identifier (ID) that is unique among all of the objects of that type, separate from its serial number (version identifier). The serial number associated with a particular ID for an object thus indicates the version of the object last downloaded to the client or last updated on the server following an upload

from a client. The examples illustrating the synchronization of data discussed above have primarily discussed synchronization of assignments, but it should clearly be understood that other types of data nodes besides assignments can be synchronized in accord with the present invention.

While the above exemplary application of the present invention is intended for use in an educational application, involving a teacher or student who must connect with the ECS on a server computer, it must be emphasized that the present invention is not in any way limited to the type of data discussed in this exemplary application. Also, it should be emphasized that the invention is not limited to a client-server paradigm, but can also be implemented on a peer-to-peer network where data on one computing device must be maintained in synchronization with data on a different computing device to which it is coupled. The network connecting the two or more computing devices in a peer-to-peer relationship can be a LAN, WAN, and/or the Internet or other public network. In addition, the computing devices are not limited to conventional personal computers and servers. Instead, other types of computing devices, such as PDAs, can be employed for coupling to another computing device for uploading or downloading data that are maintained in synchronization, in accord with the present invention.

Although the present invention has been described in connection with the preferred form of practicing it, those of ordinary skill in the art will understand that many modifications can be made thereto within the scope of the claims that follow. Accordingly, it is not intended that the scope of the invention in any way be limited by the above description, but instead be determined entirely by reference to the claims that follow.